Submitted to: 4th NASA National Symposium on Large-Scale Analysis and Design on High-Performance Computers and Workstations, Oct. 15-17, 1997, Williamsburg, VA

An Assessment of a Beowulf System for a Wide Class of Analysis and Design Software

D. S. Katz, T. Cwik, B. H. Kwan, J. Z. Lou, P. L. Springer, T. L. Sterling, and P. Wang

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Introduction

A typical Beowulf system, such as the machine at the Jet Propulsion Laboratory (JPL), may comprise 16 nodes interconnected by 100 base T Fast Ethernet. Each node may include a single Intel Pentium Pro 200 MHz microprocessor, 128 MBytes of DRAM, 2.5 GBytes of IDE disk, and PCI bus backplane, and an assortment of other devices. At least one node will have a video card, monitor, keyboard, CD-ROM, floppy drive, and so forth. But so fast is the technology evolving and price performance and price feature curve changing that no two Beowulfs ever look exactly alike. Of course, this is also because the pieces are almost always acquired for a mix of vendors and distributors. The power of *de facto* standards for interpretability of subsystems has generated an open market that provides a wealth of choices for customizing one's own version of Beowulf, or just maximizing cost advantage as prices fluctuate among sources. Such a system will run the Linux operating system freely available over the net or in low-cost and convenient CD-ROM distributions. In addition, publicly available parallel processing libraries such as MPI and PVM are used to harness the power of parallelism for large application programs. A Beowulf system such as described here, taking advantage of appropriate discounts, costs about $50K including all incidental components such as low cost packaging.

The Beowulf approach represents a new business model for acquiring computational capabilities. It complements rather than competes with the more conventional vendor-centric systems-supplier approach. Beowulf is not for everyone. Any site that would include a Beowulf cluster should have a systems administrator already involved in supporting the network of workstations and PCs the inhabit the workers' desks. Beowulf is a parallel computer, and as such, the site must be willing to run parallel programs, either developed in-house or acquired from others. Beowulf is a loosely coupled, distributed memory system, running message-passing parallel programs that do not assume a shared memory space across processors. Its long latencies require a favorable balance of computation to communication and code written to balance the workload across processing nodes. Within the constrained regime in which Beowulf is appropriate, it should provide the best performance to cost and often comparable performance per node to vendor offerings. This paper is intended to help determine where a wide variety of application codes fit within this regime.

To determine how a given message passing code will perform on a given machine, the communication characteristics of both the machine and the code must be known, as well as the computational performance of both the machine and the code. Computational performance of a

code on a Pentium Pro system is under wide study, and will not be focused upon in this paper. Rather, characterization of the communications of the codes under study and the Beowulf machine at JPL will be the main theme of this paper, with computational performance being discussed as needed.

This paper considers many application codes. The codes span a wide spectrum of communication types, in terms of message size, message frequency, and message count. These codes were chosen to represent a sampling of the types of codes used at JPL. These include codes currently run on parallel supercomputers, sequential supercomputers, workstations, and PCs. The intent of this paper is to examine the performance of a Beowulf system on these codes, in order to determine the JPL-wide usefulness of this type of machine in helping engineers and scientists at JPL do their jobs quickly and well,

Application Software

A suite of application software is considered in this assessment. It consists of a range of applications and related algorithms. There is also a range in the amount of data being communicated as well as the pattern of communication across processors. All applications use MPI or PVM for communication between processors and run on other platforms. The types and description of the applications are:

- Physical optics antenna and telescope design software; this software computes radiation patterns of antenna systems or telescopes by performing a vector two-dimensional radiation integrals of currents induced *on* antenna surfaces. The amount of computation can be intensive when the surfaces are large relative to wavelength, and the amount of communication is typically small. The communication can typically be limited to simple global sums at intermediate points of the computation whereas additional communication can develop when the geometric surface descriptions cannot be stored within a processor.

- Finite-difference time-domain electromagnetic software; this software is used for solving antenna patterns, calculating electromagnetic scattering from targets, or examining fields within small electronic circuits and boards. This version uses a uniform Cartesian grid, and describes the object being studied as a combination of cubic cells and square faces. The code uses an explicit time-marching scheme, and the parallelization is done by decomposing the physical domain being studied over the processors. At each time step, fields at the boundary of each processor's sub-domain must be passed to neighboring processors for the next time step. The communication to computation ratio is therefore related to the surface area to volume ratio of the subdomains on each processor,

- Finite-element electromagnetic software; this software is used similarly to the finite-difference software, except it returns information at certain frequencies, rather than data over a time period, The complete finite-element code builds a large sparse matrix, distributes it over the processors, and solves for many right hand sides. The building and distributing the matrix steps are completed on a workstation, and only the solution of the matrix for each of the right hand sides is done on the parallel computer. An block-like iterative scheme (quasi-minimal residual) is used for the matrix solve, in which a matrix-vector multiply is the dominant component.

- Incompressible fluid flow solver; this is a parallel implementation of a state-of-the-art numerical algorithm, a second-order projection method, for solving the incompressible

Navier-Stokes equations. Features of this algorithm include its superior numerical stability in simulating flows with high Reynolds numbers and its robustness i n resolving non-smooth, strongly sheared flows, partly due to the use of a Godunov scheme combined with an upwind scheme in the discretization of the convection term. The parallel flow solver package has *been* developed for solving two and three dimensional problems with a variety of boundary conditions on rectangular, staggered finite-difference grids. The model of the parallel implementation is domain partition and explicit message-passing. The parallel flow solver uses a parallel multigrid elliptic solver (also a stand-alone solver developed in-house at JPL) as a computation kernel to efficiently update velocity and pressure fields. A genetic message-passing interface is used in the solver package with software wrappers implemented for several message-passing libraries, including MPI, PVM and Intel NX.

- Non-linear thermal convection solver; this is a parallel implementation of the finite volume method for three-dimensional, time-dependent, thermal convective flows. The discretization equations derived from the scheme, including a pressure equation which consumes most of computation time, are solved using a parallel multigrid method. A flexible parallel code has been implemented on the distributed-memory systems, by using domain decomposition techniques and the MPI communication software. The code uses ID, 2D or 3D partitions according to different geometries. It currently runs on the Intel Paragon, the Cray T3D, and the IBM SP2, and can be easily ported to other parallel systems.

- Parallel Matlab utilities; this software allows a Matlab user to run specific operations on a parallel computer. From within Matlab running on a workstation, the user can call a parallel routine which is equivalent to a corresponding Matlab routine. The parallel code (called Matpar) initiates a session on a parallel computer, using PVM, and sends the matrix data and commands to software residing on the parallel computer. The operation is executed in parallel, and the result sent back to Matlab in standard Matlab format.

Assessment Parameters and Purpose

The application codes will be described by how they use communication and computation. Key parameters are: number of floating point operations, total number of operations, number of communication calls, frequency of communications calls, and length of communication calls. The Beowulf system will be described by its communication performance, including data showing time required to send various size messages among various numbers of processors. The intent of this paper is to discuss how a description of a machine and a description of an application code can be used to predict performance of the code on that machine, and to discuss what this performance implies about the feasibility of using a Beowulf class machine to run these science and engineering codes in an institutional environment, such as at JPL.